IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Applicant:	Wolczko et al.	
App. No.:	10/780,264	Conf. No.: 2218
Filed:	February 16, 2004	Art Unit: 2192
Title:	INSTRUCTION SAMPLING IN A MULTI-THREADED PROCESSOR	Examiner: Yigdall, Michael J.

APPEAL BRIEF

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Applicant (hereafter "Appellant") hereby submits this Appeal Brief in response to the to the final Office action mailed August 4, 2009 and the Notice of Appeal filed on December 4, 2009 in the above-captioned case.

Appellant respectfully requests consideration of this appeal by the Board of Patent Appeals and Interferences (hereafter the "Board").

An oral hearing is not requested at this time.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The Appellant is Sun Microsystems, Inc. of 4150 Network Circle, Santa Clara, California 95054.

II. RELATED APPEALS AND INTERFERENCES

To the best of Appellant's knowledge, there are no appeals or interferences that are related to, will directly affect, will be directly affected by, or have a bearing on the Board's decision in the present appeal.

III. STATUS OF THE CLAIMS

Claims 1-18 and 20-23 are currently pending in the above-referenced application. The rejections of all of the pending claims are appealed herein. No claims have been allowed. Claim 19 is cancelled. A clean copy of all claims on appeal is attached hereto as the Appendix of Claims. We note that previously cancelled claims are not provided in the Appendix.

IV. STATUS OF AMENDMENTS

The last response submitted by the Appellant was dated May 4, 2009, in which claims 8, 14 and 18 were amended in response to the Office action dated February 3, 2009. A final Office action was entered August 4, 2009, to which no further response or amendment was submitted by the Appellant. A Notice of Appeal was filed on December 4, 2009.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Independent claim 1 provides for a method for sampling instructions that are executing in a multi-threaded processor. More particularly, independent claim 1 recites that, when it is determined that the sampling information includes an event of interest, the sampling mechanism may then report gathered sampled information to a particular thread of the multi-threaded processor within which the instruction is executed. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* The method of claim 1 may be performed by a sampling mechanism of a processor. *See, e.g., specification, page 5, lines 8-9 and Figure 1.* The method may begin when the sampling mechanism selects an instruction executing on the multi-threaded processor for sampling. *See, e.g., Figure 2A, reference number 224 and specification, page 5, lines 13-14.* Next, information relating to the sampled instruction is captured and stored during execution of the instruction by the processor. *See, e.g., Figure 2A, reference number 234 and specification, page 5, lines 21-23.* Such information may include events induced by the instruction and associated latencies for the sampled instruction. *See, e.g., specification, page 5, line 21 and page 4, lines 16-17.* Next, the sampling mechanism determines if the sampled information includes an event of interest to a particular thread of the multi-threaded processor within which the instruction is executed. *See, e.g., Figure 2A, reference number 240 and specification, page 5, lines 27-31.* For example, in one embodiment, software indicates which properties or events are of interest to a particular thread via a bit vector contained in a register. *See, e.g., specification, page 4, lines 21-23.* The hardware mechanism that gathers the sampled instructions may then compare the samples against the bit vector to determine if the sample information includes an event of interest. *See, e.g., specification, page 4, lines 25-28.* When it is determined that the sampling information includes an event of interest, the sampling mechanism reports the gathered sampled information to the particular thread of the multi-threaded processor within which the instruction is executed. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* Once reported, a software performance monitoring tool may use the reported information to analyze the performance of the multi-threaded processor. *See, e.g., specification, page 1, lines 12-16.*

Independent claim 8 also provides for a method for sampling instructions that are executing in a multi-threaded processor. More particularly, independent claim 8 recites that, when it is determined that a candidate counter is zero, a sampling mechanism reports the sampled instruction to a particular thread of the multi-threaded processor from which the instruction is sampled. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25* . Similar to independent claim 1, one or

more of the operations of the method may be performed by a sampling mechanism of a processor. *See, e.g., specification, page 5, lines 8-9 and Figure 1.* The method may begin when software sets a candidate counter register with a number, such as a non-zero value. *See, e.g., Figure 2A, reference number 210 and specification, page 5, lines 9-11.* Next, the sampling mechanism selects an instruction executing on the multi-threaded processor for sampling. *See, e.g., Figure 2A, reference number 224 and specification, page 5, lines 13-14.* Information relating to the sampled instruction is then captured and stored during execution of the instruction by the processor. *See, e.g., Figure 2A, reference number 234 and specification, page 5, lines 21-23.* Such information may include events induced by the instruction and associated latencies for the sampled instruction. *See, e.g., specification, page 5, line 21 and page 4, lines 16-17.* Next, the sampling mechanism determines whether all events for the instruction have occurred. *See, e.g., Figure 2A, reference number 236 and specification, page 5, lines 23-25.* If it is determined that all events for the instruction have occurred and the sampled instruction corresponds to a desired sampled thread, the sampling mechanism may decrement the candidate counter and determine if the candidate counter is zero. *See, e.g., Figure 2A, reference numbers 244 & 246 and specification, page 6, lines 3-5.* When it is determined that the candidate counter is zero, the sampling mechanism reports the sampled instruction to a particular thread of the multi-threaded processor from which the instruction is sampled. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* Once reported, a software performance monitoring tool may use the reported information to analyze the performance of the multi-threaded processor. *See, e.g., specification, page 1, lines 12-16.*

Independent claim 14 also provides for a method for sampling instructions that are executing in a multi-threaded processor. More particularly, when it is determined that a candidate counter is zero, the sampling mechanism reports the sampled instruction to a particular thread of the multi-threaded processor from which the instruction is sampled. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* Similar to independent claims 1 and 8, one or more of the operations of the method may be performed by a sampling mechanism of a processor. *See, e.g., specification, page 5, lines 8-9 and Figure 1.* The method may begin when software sets a candidate counter register with a number, such as a non-zero value. *See, e.g., Figure 2A, reference number 210 and specification, page 5, lines 9-11.* Next, the sampling mechanism selects an instruction executing on the multi-threaded processor for sampling. *See, e.g., Figure 2A, reference number 224 and specification, page 5, lines 13-14.* Information relating to the sampled instruction is then captured and stored during execution of the instruction by the processor. *See, e.g., Figure 2A, reference number 234 and specification, page 5, lines 21-23.* Such information may include events induced by the instruction and associated

latencies for the sampled instruction. *See, e.g., specification, page 5, line 21 and page 4, lines 16-17.* Next, the sampling mechanism determines if the sampled information includes events of interest, including whether the sampled instruction corresponds to a particular sampled thread of the multi-threaded processor. *See, e.g., Figure 2A, reference number 240 and specification, page 5, lines 27-31.* If it is determined that all events for the instruction have occurred and the sampled instruction includes events of interest, the sampling mechanism may decrement the candidate counter and determine if the candidate counter is zero. *See, e.g., Figure 2A, reference numbers 244 & 246 and specification, page 6, lines 3-5.* When it is determined that the candidate counter is zero, the sampling mechanism may then report the sampled instruction to a particular thread of the multi-threaded processor from which the instruction is sampled. *See, e.g., Figure 2B, reference number 248; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* Once reported, a software performance monitoring tool may use the reported information to analyze the performance of the multi-threaded processor. *See, e.g., specification, page 1, lines 12-16.*

Independent claim 18 provides for a multi-threaded processor. The multi-threaded processor includes sampling logic to determine whether an instruction executed in the processor corresponds to a desired thread sample. *See, e.g., Figure 1, reference number 120 and specification, page 5, lines 27-31.* The multi-threaded processor also includes a sampling register logic coupled to the sampling logic. *See, e.g., Figure 1, reference number 124.* An instruction history register logic is coupled to the sampling register logic and configured to store information relating to the instruction. *See, e.g., Figure 1, reference number 122 and specification, page 4, lines 16-17.* Such information may include events induced by the instruction and associated latencies for the instruction. *See, e.g., specification, page 5, line 21 and page 4, lines 16-17.* The multi-threaded processor also includes a sample filtering and counting logic that is coupled to the sampling logic and replicated on a per-thread basis for each thread of the multi-threaded processor. *See, e.g., Figure 1, reference number 126 and specification, page 6, lines 20-21.* In addition, the multi-threaded processor includes notification logic configured to report the gathered sampled information to the particular thread of the multi-threaded processor within which the instruction is executed if the instruction corresponds to the desired sampled thread. *See, e.g., Figure 1, reference number 128; specification, page 6, lines 7-8; and specification, page 6, lines 23-25.* Once reported, a software performance monitoring tool may use the reported information to analyze the performance of the multi-threaded processor. *See, e.g., specification, page 1, lines 12-16.*

VI. GROUNDS OF REJECTION PRESENTED FOR REVIEW

A. Whether independent claims 1, 8, 14 and 18 (and related dependent claims) are unpatentable under 35 U.S.C. § 103(a) over U.S. Patent No. 6,000,044 to Chrysos et al. (hereafter "Chrysos") in view of U.S. Patent No. 7,448,025 to Kalafatis et al. (hereafter "Kalafatis").

VII. ARGUMENT

A. THE REJECTION OF INDEPENDENT CLAIMS 1, 8, 14 AND 18 AND THE RELATED DEPENDENT CLAIMS UNDER 35 U.S.C. § 103(a) IS IMPROPER BECAUSE THE COMBINED REFERENCES FAIL TO DISCLOSE ALL OF THE ELEMENTS OF THE REJECTED CLAIMS

Claims 1-18 and 20-23 are rejected under 35 U.S.C. § 103 as being unpatentable over U.S. Patent No. 6,000,044 to Chrysos et al. (hereafter "Chrysos") in view of U.S. Patent No. 7,448,025 to Kalafatis et al. (hereafter "Kalafatis"). Claims 1, 8, 14 and 18 are independent claims and the remaining claims depend therefrom. The basis of rejection of the dependent claims stems from the rejection of the independent claim; accordingly, the rejection of independent claims 1, 8, 14 and 18 is considered first.

A proper obviousness rejection based on a rationale of combining prior art elements requires at least "a finding that the prior art included each element claimed . . . with the only difference between the claimed invention and the prior art being the lack of actual combination of the elements." MPEP § 2143(A). *See also In re Royka*, 490 F.2d 981, 985 (C.C.P.A. 1974); *CFMT, Inc. v. YieldUp Int'l Corp.*, 349 F.3d 1333, 1342 (Fed. Cir. 2003). In other words, to establish a proper case of obviousness of the claimed invention, all of the claim limitations must be taught or suggested by the prior art. Appellant respectfully suggests that the cited references fail to teach or suggest all of the limitations of independent claims 1, 8, 14 and 18 and, thus, a proper case of obviousness has not been established.

1. Independent claims 1, 8, 14 and 18 are patentable because neither Chrysos nor Kalafatis teach or suggest reporting information about a sampled instruction to a particular thread when the information includes an event of interest.

Independent claim 1 includes the limitations of storing sampling information related to a sampled instruction, determining whether the stored information includes an event of interest to a particular thread and "reporting the sampling information to the particular thread when the sampling information includes an event of interest." Similarly, independent claims 8 and 14 include the limitations of setting a candidate counter and "reporting the instruction to a particular thread when the candidate counter equals zero". Also, independent claim 18 includes a notification logic for "reporting to a particular thread the information relating to the instruction if the instruction corresponds to the desired sampled thread." Appellant respectfully submits that neither Chrysos nor Kalafatis teach or suggest these limitations such that the combination of Chrysos in view of Kalafatis fails to render obvious independent claims 1, 8, 14 and 18.

Chrysos discloses an apparatus for randomly sampling instructions in a processor pipeline. *See Chrysos, Abstract.* However, as correctly recognized by the Examiner in the final Office action dated August 4, 2009, Chrysos does not disclose reporting the sampling information or the instruction to a particular thread as required by the independent claims.

See final Office action dated 08/04/09, pages 5, 8, 11 and 13. Rather, the final Office action relies on Kalafatis to disclose this limitation of the independent claims. *See final Office action dated 08/04/09, pages 5, 8, 11 and 13.*

Not only does Chrysos fail to disclose reporting the sampling information to a particular thread, the disclosure of Chrysos fails to teach or suggest any type of reporting of a sampled instruction. Rather, the apparatus of Chrysos only samples an instruction and stores information related to the execution of the instruction. The stored information is then available for access by a software program. *See Chrysos, col. 17, lines 35-39.* In other words, Chrysos discloses sampling and storing, but fails to disclose reporting any stored information, let alone reporting to a particular thread. In contrast, the independent claims of the present application include the limitation of reporting the gathered information or the instruction to a particular thread. Thus, because the apparatus of Chrysos only stores the gathered information and does not disclose reporting such information to a particular thread, Chrysos fails to disclose this limitation of the independent claims of the present application.

Similarly, there is simply no teaching or suggestion in Kalafatis of reporting an instruction or information related to a sampled instruction to any particular thread. Kalafatis discloses an apparatus for monitoring performance of a multithreaded processor by detecting specific processor events. *See Kalafatis, Abstract.* Events that qualify are counted in an event counter that can be accessed by a program for processing. *See Kalafatis, Abstract.* However, similar to Chrysos, the apparatus of Kalafatis fails to disclose reporting the sampling information or the instruction to a particular thread as required by the independent claims. Instead, Kalafatis teaches that external software may access and sample the contents of event counters via a program instruction in order to process the stored information. *See Kalafatis, col. 2 lines 52-54; col. 6 lines 29-39 ("system software can program/sample the contents of each of the registers").* Thus, the apparatus of Kalafatis operates in a similar manner as described above in relation to the apparatus of Chrysos in that information gathered about a sampled instruction is stored and made available to a software program, but not reported to a thread.

Like Chrysos, the apparatus of Kalafatis requires an additional software program to access the stored information rather than having the information reported to the particular thread. Thus, because the apparatus of Kalafatis operates similarly to that of Chrysos in that it only stores the gathered information and does not report the information to a particular thread, Kalatafis also fails to disclose this limitation of the independent claims of the present application.

2. Conclusion

As shown above, both Chrysos and Kalafatis fail to disclose the limitation of reporting the sampling information or the instruction to a particular thread as required by the

independent claims. Therefore, a combination of Chrysos in view of Kalafatis would likewise fail to teach or suggest the recited limitation. Thus, for at least the reasons recited above, it is respectfully submitted that Chrysos in view of Kalafatis fails to disclose all of the limitations of independent claims 1, 8, 14 and 18 and the respective dependent claims, and thus these claims are patentable under 35 U.S.C. § 103 over the cited combination.

B. ALL OTHER REJECTED DEPENDENT CLAIMS ARE PATENTABLE OVER THE RECITED COMBINATIONS OF PRIOR ART

All additional rejections to dependent claims 2-7, 9-13, 15-17 and 20-23 are based on the combination of Chrysos in view of Kalafatis. Accordingly, it is respectfully submitted that these dependent claims are patentable over the combination of Chrysos in view of Kalafatis for at least the same reasons as independent claims, 1, 8, 14 and 18.

VIII. CONCLUSION

Appellant respectfully submits that all the appealed claims in this application are patentable and requests that the Board of Patent Appeals and Interferences direct allowance of the rejected claims.

Appellant believes no further fees or petitions are due with this filing. However, if any such petitions or fees are necessary, please consider this a request therefor and authorization to charge Deposit Account No. 04-1415 accordingly.

Dated: Feb. 4, 2010

Respectfully submitted,

Gregory P. Durbin, Registration No. 42,503
Attorney for Applicant
USPTO Customer No. 66083

DORSEY & WHITNEY LLP
Republic Plaza Building, Suite 4700
370 Seventeenth Street
Denver, Colorado 80202-5647
Phone: (303) 629-3400
Fax: (303) 629-3450

IX. APPENDIX OF CLAIMS

1. A method of sampling instructions executing in a multi-threaded processor comprising:

selecting an instruction for sampling;

storing sampling information relating to the instruction;

determining whether the sampling information includes an event of interest to a particular thread within which the instruction is executing; and

reporting the sampling information to the particular thread when the sampling information includes an event of interest.

2. The method of claim 1 further comprising providing a register with a bit vector representing a plurality of events of interest; and

wherein the determining whether the sampling information includes the event of interest further includes comparing the sampling information relating to the instruction to the bit vector.

3. The method of claim 2 wherein the comparing is via at least one of a mask operation or a more expressive operation.

4. The method of claim 1 wherein the selecting the instruction is without regard to a thread to which the instruction is bound.

5. The method of claim 1 further comprising identifying a thread to which the instruction is bound when the instruction is selected.

6. The method of claim 1 further comprising providing filtering criteria on a per-thread basis.

7. The method of claim 1 further comprising providing a single set of filtering criteria; and, scheduling sampling among a plurality of threads via software.

8. A method of sampling instructions executing in a multi-threaded processor comprising:

setting a candidate counter to a number;

selecting an instruction for sampling;

storing information relating to the instruction;

determining whether all events for the instruction have occurred;

decrementing the candidate counter when all events for the instruction have occurred and when the instruction corresponds to a desired sampled thread;

determining whether the candidate counter equals zero; and

reporting the instruction to a particular thread when the candidate counter equals zero.

9. The method of claim 8 wherein the information relating to the instruction represents an instruction history, and the instruction history includes information relating to at least one of an events value, a program counter value, a branch target address value, an effective memory address value, a latency value, a number in issue bundle value, a number in retire bundle value, a privilege value, a branch history value and a number in fetch bundle value.

10. The method of claim 8 wherein the selecting the instruction is without regard to a thread to which the instruction is bound.

11. The method of claim 8 further comprising identifying a thread to which the instruction is bound when the instruction is selected.

12. The method of claim 8 further comprising providing filtering criteria on a per-thread basis.

13. The method of claim 8 further comprising providing a single set of filtering criteria; and, scheduling sampling among a plurality of threads via software.

14. A method of sampling instructions executing in a multi-threaded processor comprising:

setting a candidate counter to a number;

selecting an instruction for sampling;

storing information relating to the instruction;

determining whether all events for the instruction have occurred;

determining whether the instruction includes events of interest, the events of interest including whether the instruction corresponds to a desired sampled thread;

decrementing the candidate counter when all events for the instruction have occurred and when the instruction includes events of interest;

determining whether the candidate counter equals zero; and

reporting the instruction to a particular thread when the candidate counter equals zero.

15. The method of claim 14 further comprising providing a register with a bit vector representing events of interest; and

wherein the determining whether the instruction includes events of interest further includes comparing the information relating to the instruction to the bit vector.

16. The method of claim 14 wherein the information relating to the instruction represents an instruction history, and the instruction history includes information relating to at least one of an event value, a program counter value, a branch target address value, an effective memory address value, a latency value, a number in issue bundle value, a number in retire bundle value, a privileged value, a branch history value and a number in fetch bundle value.

17. The method of claim 14 wherein the selecting an instruction for sampling is based upon sample selection criteria; and

the sample selection criteria include information relating to a desired sampled thread.

18. A multi-threaded processor comprising:

a sampling logic configured to determine whether an instruction executed in the processor corresponds to a desired sampled thread;

a sampling register logic coupled to the sampling logic;

an instruction history register logic coupled to the sampling register logic, the instruction history register logic storing information relating to the instruction;

a sample filtering and counting logic coupled to the sampling logic,

wherein the sample filtering and counting logic is replicated on a per thread basis; and

a notification logic, the notification logic reporting to a particular thread the information relating to the instruction if the instruction corresponds to the desired sampled thread.

20. The processor of claim 18 wherein the sampling register logic includes a register with a bit vector representing events of interest; and

wherein the sampling logic determines whether the instruction includes events of interest by comparing the information relating to the instruction to the bit vector.

21. The processor of claim 18 wherein the information relating to the instruction represents an instruction history, and the instruction history includes information relating to at least one of an events value, a program counter value, a branch target address value, an effective memory address value, a latency value, a number in issue bundle value, a number in retire bundle value, a privileged value, a branch history value and a number in fetch bundle value.

22. The processor of claim 18 wherein the sampling register logic includes a sample selection criteria register storing sample selection criteria; and the sample selection criteria include information relating to a desired sampled thread.

23. The method of claim 1 wherein storing the sampling information further comprises storing the sampling information to a memory register shared by multiple threads.

X. EVIDENCE APPENDIX

None.

XI. RELATED PROCEEDINGS APPENDIX

None.